


GALAHAD HASH

USER DOCUMENTATION GALAHAD Optimization Library version 5.1

1 SUMMARY

This package **sets up a dictionary of words, allowing a user to insert new words, and search for and remove existing words**. It also allows the user to rebuild the dictionary if the maximum allowed word-size, or the total space provided, proves too small. Provided sufficient room is allowed, the expected number of operations required for an insertion, search or removal is $O(1)$. The method is based on the chained scatter table insertion method of F. A. Williams.

ATTRIBUTES — Versions: `GALAHAD_HASH_single, GALAHAD_HASH_double`. **Uses:** `GALAHAD_SYMBOLS`, `GALAHAD_SPECFILE` and `GALAHAD_SPACE`. **Date:** July 2021. **Origin:** N. I. M. Gould, Rutherford Appleton Laboratory. **Language:** Fortran 95 + TR 15581 or Fortran 2003.

2 HOW TO USE THE PACKAGE

The package is available with single, double and (if available) quadruple precision reals, and either 32-bit or 64-bit integers. Access to the 32-bit integer, single precision version requires the `USE` statement

```
USE GALAHAD_HASH_single
```

with the obvious substitution `GALAHAD_HASH_double`, `GALAHAD_HASH_quadruple`, `GALAHAD_HASH_single_64`, `GALAHAD_HASH_double_64` and `GALAHAD_HASH_quadruple_64` for the other variants.

If it is required to use more than one of the modules at the same time, the derived types `HASH_time_type`, `HASH_control_type`, `HASH_inform_type`, `HASH_data_type` and (Section 2.2) and the subroutines `HASH_initialize`, `HASH_insert`, `HASH_search`, `HASH_remove`, `HASH_rebuild` `HASH_terminate`, (Section 2.3) and `HASH_read_specfile` (Section 2.5) must be renamed on one of the `USE` statements.

2.1 Real and integer kinds

We use the terms integer and real to refer to the fortran keywords `REAL(rp_)` and `INTEGER(ip_)`, where `rp_` and `ip_` are the relevant kind values for the real and integer types employed by the particular module in use. The former are equivalent to default `REAL` for the single precision versions, `DOUBLE PRECISION` for the double precision cases and quadruple-precision if 128-bit reals are available, and correspond to `rp_ = real32`, `rp_ = real64` and `rp_ = real128` respectively as defined by the fortran `iso_fortran_env` module. The latter are default (32-bit) and long (64-bit) integers, and correspond to `ip_ = int32` and `ip_ = int64`, respectively, again from the `iso_fortran_env` module.

2.2 The derived data types

Three derived data types are accessible from the package.

2.2.1 The derived data type for holding control parameters

The derived data type `HASH_control_type` is used to hold controlling data. Default values may be obtained by calling `HASH_initialize` (see Section 2.3.1), while components may also be changed by calling `GALAHAD_HASH_read_spec` (see Section 2.5.1). The components of `HASH_control_type` are:

error is a scalar variable of type INTEGER(ip_), that holds the stream number for error messages. Printing of error messages in HASH_solve and HASH_terminate is suppressed if error ≤ 0. The default is error = 6.

out is a scalar variable of type INTEGER(ip_), that holds the stream number for informational messages. Printing of informational messages in HASH_solve is suppressed if out < 0. The default is out = 6.

print_level is a scalar variable of type INTEGER(ip_), that is used to control the amount of informational output which is required. No informational output will occur if print_level ≤ 0. If print_level $= 1$, a single line of output will be produced for each iteration of the process. If print_level ≥ 2, this output will be increased to provide significant detail of each iteration. The default is print_level = 0.

space_critical is a scalar variable of type default LOGICAL, that must be set .TRUE. if space is critical when allocating arrays and .FALSE. otherwise. The package may run faster if space_critical is .FALSE. but at the possible expense of a larger storage requirement. The default is space_critical = .FALSE..

deallocate_error_fatal is a scalar variable of type default LOGICAL, that must be set .TRUE. if the user wishes to terminate execution if a deallocation fails, and .FALSE. if an attempt to continue will be made. The default is deallocate_error_fatal = .FALSE..

prefix is a scalar variable of type default CHARACTER and length 30, that may be used to provide a user-selected character string to preface every line of printed output. Specifically, each line of output will be prefaced by the string prefix(2:LEN(TRIM(prefix))-1), thus ignoreing the first and last non-null components of the supplied string. If the user does not want to preface lines by such a string, they may use the default prefix = "".

2.2.2 The derived data type for holding informational parameters

The derived data type HASH_inform_type is used to hold parameters that give information about the progress and needs of the algorithm. The components of HASH_inform_type are:

status is a scalar variable of type INTEGER(ip_), that gives the exit status of the algorithm. See Section 2.4 for details.

alloc_status is a scalar variable of type INTEGER(ip_), that gives the status of the last attempted array allocation or deallocation. This will be 0 if status = 0.

bad_alloc is a scalar variable of type default CHARACTER and length 80, that gives the name of the last internal array for which there were allocation or deallocation errors. This will be the null string if status = 0.

2.2.3 The derived data type for holding problem data

The derived data type HASH_data_type is used to hold all the data associated with the dictionary built and maintained between calls of HASH procedures. This data should be preserved, untouched from the initial call to HASH_initialize to the final call to HASH_terminate.

2.3 Argument lists and calling sequences

All the words in the dictionary are entered into a so-called *chained scatter table*. Before the first word is entered, the table must be initialized by a call to HASH_initialize to set default values, and initialize private data. Words are inserted in the table by calling HASH_insert. The table may be searched for an existing word with a call to HASH_search; an existing word may be flagged as deleted from the table by calling HASH_remove. Finally, the table may be rebuilt to allow for an increase in the maximum allowed word-size or the total number of entries in the table with a call to HASH_rebuild. All internally allocated workspace may be removed by calling HASH_terminate.

2.3.1 The initialization subroutine

Dictionary initialization and default control parameters are provided as follows:

```
CALL HASH_initialize( nchar, length, data, control, inform )
```

nchar is a scalar INTENT(IN) argument of type INTEGER(ip_) that should be sent to an upper bound on the number of characters in each word that may be inserted into the dictionary.

length is a scalar INTENT(IN) argument of type INTEGER(ip_) that should be sent to an upper bound on the number of words that may be inserted into the dictionary.

data is a scalar INTENT(INOUT) argument of type HASH_data_type (see Section 2.2.3). It is used to hold data about the dictionary.

control is a scalar INTENT(OUT) argument of type HASH_control_type (see Section 2.2.1). On exit, control contains default values for the components as described in Section 2.2.1. These values should only be changed after calling HASH_initialize.

inform is a scalar INTENT(OUT) argument of type HASH_inform_type (see Section 2.2.2). A successful call to HASH_initialize is indicated when the component status has the value 0. For other return values of status, see Section 2.4.

2.3.2 The insertion subroutine

A word may be inserted into the dictionary as follows:

```
CALL HASH_insert( nchar, field, position, data, control, inform )
```

nchar is a scalar INTENT(IN) argument of type INTEGER(ip_) that should be sent to an upper bound on the number of characters in each word that may be inserted into the dictionary.

field is an array INTENT(IN) argument of length nchar and type default CHARACTER that contains the characters of the word that is to be inserted into the dictionary. Component field(j) should be filled with the j-th character of the word. If the word contains fewer than nchar characters, it should be padded with blanks.

position is a scalar INTENT(OUT) argument of type INTEGER(ip_) that gives the index of the table that data for the word occupies after insertion. If position=0 on exit, there is no more room in the dictionary, and it should be rebuilt (see §2.3.5) with more space before trying the insertion again.

data is a scalar INTENT(INOUT) argument of type HASH_data_type (see Section 2.2.3). It is used to hold data about the dictionary.

control is a scalar INTENT(IN) argument of type HASH_control_type (see Section 2.2.1). On exit, control contains default values for the components as described in Section 2.2.1. These values should only be changed after calling HASH_initialize.

inform is a scalar INTENT(INOUT) argument of type HASH_inform_type (see Section 2.2.2). A successful call to HASH_initialize is indicated when the component status has the value 0. For other return values of status, see Section 2.4.

2.3.3 The search subroutine

A word may be searched for in the dictionary as follows:

```
CALL HASH_search( nchar, field, position, data, control, inform )
```

nchar is a scalar INTENT(IN) argument of type INTEGER(ip_) that should be sent to an upper bound on the number of characters in each word that may be inserted into the dictionary.

field is an array INTENT(IN) argument of length nchar and type default CHARACTER that contains the characters of the word that is to be searched for in the dictionary. Component field(j) should be filled with the j-th character of the word. If the word contains fewer than nchar characters, it should be padded with blanks.

position is a scalar INTENT(OUT) argument of type INTEGER(ip_) that gives the index of the table that data for the word occupies. If the word is not found, position will be 0, and if the word has been removed, it will be negative (and - position was the index that it once occupied).

data is a scalar INTENT(INOUT) argument of type HASH_data_type (see Section 2.2.3). It is used to hold data about the dictionary.

control is a scalar INTENT(IN) argument of type HASH_control_type (see Section 2.2.1). On exit, control contains default values for the components as described in Section 2.2.1. These values should only be changed after calling HASH_initialize.

inform is a scalar INTENT(INOUT) argument of type HASH_inform_type (see Section 2.2.2). A successful call to HASH_initialize is indicated when the component status has the value 0. For other return values of status, see Section 2.4.

2.3.4 The removal subroutine

A word may be removed from the dictionary as follows:

```
CALL HASH_remove( nchar, field, position, data, control, inform )
```

nchar is a scalar INTENT(IN) argument of type INTEGER(ip_) that should be sent to an upper bound on the number of characters in each word that may be inserted into the dictionary.

field is an array INTENT(IN) argument of length nchar and type default CHARACTER that contains the characters of the word that is to be inserted into the dictionary. Component field(j) should be filled with the j-th character of the word. If the word contains fewer than nchar characters, it should be padded with blanks.

position is a scalar INTENT(OUT) argument of type INTEGER(ip_) that gives the index of the table that data for the word occupies before it was removed. If the word is not found, position will be 0.

data is a scalar INTENT(INOUT) argument of type HASH_data_type (see Section 2.2.3). It is used to hold data about the dictionary.

control is a scalar INTENT(IN) argument of type HASH_control_type (see Section 2.2.1). On exit, control contains default values for the components as described in Section 2.2.1. These values should only be changed after calling HASH_initialize.

inform is a scalar INTENT(INOUT) argument of type HASH_inform_type (see Section 2.2.2). A successful call to HASH_initialize is indicated when the component status has the value 0. For other return values of status, see Section 2.4.

2.3.5 The rebuilding subroutine

The dictionary may be rebuilt to increase in its length as follows:

```
CALL HASH_rebuild( length, new_length, moved_to, data, control, inform )
```

length is a scalar INTENT(IN) argument of type INTEGER(ip_) that should be the current length of the dictionary.

new_length is a scalar INTENT(IN) argument of type INTEGER(ip_) that should be the new length of the dictionary.

moved_to is an array INTENT(OUT) argument of length length and type INTEGER(ip_) that gives the position in the new table that the old table entry has been moved to. Specifically, if moved_to(i) is nonzero, entry i has moved to position moved_to(i) in the new dictionary, while if moved_to(i)=0, entry i was not previously occupied.

data is a scalar INTENT(INOUT) argument of type HASH_data_type (see Section 2.2.3). It is used to hold data about the dictionary.

control is a scalar INTENT(IN) argument of type HASH_control_type (see Section 2.2.1). On exit, control contains default values for the components as described in Section 2.2.1. These values should only be changed after calling HASH_initialize.

inform is a scalar INTENT(INOUT) argument of type HASH_inform_type (see Section 2.2.2). A successful call to HASH_initialize is indicated when the component status has the value 0. For other return values of status, see Section 2.4.

2.3.6 The termination subroutine

All previously allocated arrays are deallocated as follows:

```
CALL HASH_terminate( data, control, inform )
```

data is a scalar INTENT(INOUT) argument of type HASH_data_type exactly as for HASH_solve, which must not have been altered **by the user** since the last call to HASH_initialize. On exit, array components will have been deallocated.

control is a scalar INTENT(IN) argument of type HASH_control_type exactly as for HASH_solve.

inform is a scalar INTENT(OUT) argument of type HASH_inform_type exactly as for HASH_solve. Only the component status will be set on exit, and a successful call to HASH_terminate is indicated when this component status has the value 0. For other return values of status, see Section 2.4.

2.4 Warning and error messages

A negative value of inform%status on exit from HASH_solve or HASH_terminate indicates that an error has occurred. No further calls should be made until the error has been corrected. Possible values are:

-1. A workspace allocation error occurred. A message indicating the offending array is written on unit control%error, and the returned allocation status and a string containing the name of the offending array are held in inform%alloc_status and inform%bad_alloc, respectively.

-2. A workspace deallocation error occurred. A message indicating the offending array is written on unit control%error and the returned allocation status and a string containing the name of the offending array are held in inform%alloc_status and inform%bad_alloc, respectively.

-99. The current dictionary is full and should be rebuilt with more space (see §2.3.5).

2.5 Further features

In this section, we describe an alternative means of setting control parameters, that is components of the variable control of type HASH_control_type (see Section 2.2.1), by reading an appropriate data specification file using the subroutine HASH_read_specfile. This facility is useful as it allows a user to change HASH control parameters without editing and recompiling programs that call HASH.

A specification file, or specfile, is a data file containing a number of "specification commands". Each command occurs on a separate line, and comprises a "keyword", which is a string (in a close-to-natural language) used to identify a control parameter, and an (optional) "value", which defines the value to be assigned to the given control parameter. All keywords and values are case insensitive, keywords may be preceded by one or more blanks but values must not contain blanks, and each value must be separated from its keyword by at least one blank. Values must not contain more than 30 characters, and each line of the specfile is limited to 80 characters, including the blanks separating keyword and value.

The portion of the specification file used by HASH_read_specfile must start with a "BEGIN HASH" command and end with an "END" command. The syntax of the specfile is thus defined as follows:

```
( .. lines ignored by HASH_read_specfile .. )
  BEGIN HASH
     keyword    value
     .......    .....
     keyword    value
  END
( .. lines ignored by HASH_read_specfile .. )
```

where keyword and value are two strings separated by (at least) one blank. The "BEGIN HASH" and "END" delimiter command lines may contain additional (trailing) strings so long as such strings are separated by one or more blanks, so that lines such as

```
BEGIN HASH SPECIFICATION
```

and

```
END HASH SPECIFICATION
```

are acceptable. Furthermore, between the "BEGIN HASH" and "END" delimiters, specification commands may occur in any order. Blank lines and lines whose first non-blank character is ! or * are ignored. The content of a line after a ! or * character is also ignored (as is the ! or * character itself). This provides an easy manner to "comment out" some specification commands, or to comment specific values of certain control parameters.

The value of a control parameters may be of three different types, namely integer, logical or real. Integer and real values may be expressed in any relevant Fortran integer and floating-point formats (respectively). Permitted values for logical parameters are "ON", "TRUE", ".TRUE.", "T", "YES", "Y", or "OFF", "NO", "N", "FALSE", ".FALSE." and "F". Empty values are also allowed for logical control parameters, and are interpreted as "TRUE".

The specification file must be open for input when HASH_read_specfile is called, and the associated device number passed to the routine in device (see below). Note that the corresponding file is REWINDed, which makes it possible to combine the specifications for more than one program/routine. For the same reason, the file is not closed by HASH_read_specfile.

2.5.1 To read control parameters from a specification file

Control parameters may be read from a file as follows:

```
CALL HASH_read_specfile( control, device )
```

control is a scalar INTENT(INOUT) argument of type HASH_control_type (see Section 2.2.1). Default values should have already been set, perhaps by calling HASH_initialize. On exit, individual components of control may have been changed according to the commands found in the specfile. Specfile commands and the component (see Section 2.2.1) of control that each affects are given in Table 2.1 on the facing page.

device is a scalar INTENT(IN) argument of type INTEGER(ip_), that must be set to the unit number on which the specfile has been opened. If device is not open, control will not be altered and execution will continue, but an error message will be printed on unit control%error.

command	component of control	value type
error-printout-device	%error	integer
printout-device	%out	integer
print-level	%print_level	integer
space-critical	%space_critical	logical
deallocate-error-fatal	%deallocate_error_fatal	logical

Table 2.1: Specfile commands and associated components of control.

2.6 Information printed

If control%print_level is positive, basic information about the progress of the algorithm will be printed on unit control%out.

3 GENERAL INFORMATION

Use of common: None.

Workspace: Provided automatically by the module.

Other routines called directly: None.

Other modules used directly: HASH_solve calls the GALAHAD packages GALAHAD_SYMBOLS, GALAHAD_SPECFILE and GALAHAD_SPACE.

Input/output: Output is under control of the arguments control%error, control%out and control%print_level.

Restrictions: None.

Portability: ISO Fortran 95 + TR 15581 or Fortran 2003. The package is thread-safe.

4 METHOD

To insert a word in the table, the word is first mapped onto an integer value, the entry integer. This mapping is often called hashing. As many words may be mapped to the same value (a collision), a chain of used locations starting from the entry integer is searched until an empty location is found. The word is inserted in the table at this point and the chain extended to the next unoccupied entry. The hashing routine is intended to reduce the number of collisions. Words are located and flagged as deleted from the table in exactly the same way; the word is hashed and the resulting chain searched until the word is matched or the end of the chain reached. Provided there is sufficient space in the table, the expected number of operations needed to perform an insertion, search or removal is $O(1)$.

References:

The chained scatter table search and insertion method is due to

F. A. Williams (1959), "Handling identifies as internal symbols in language processors", Communications of the ACM **2(6)** pp 21-24.

5 EXAMPLES OF USE

As a simple example, we read a set of words into a table and then test if a second set of words is present. Each set is terminated by a blank. A maximum word length of 10 characters is specified. We may use the following code.

```
! THIS VERSION: GALAHAD 4.1 - 2022-11-25 AT 09:00 GMT.
   PROGRAM GALAHAD_HASH_EXAMPLE
   USE GALAHAD_HASH
   IMPLICIT NONE
   TYPE ( HASH_data_type ) :: data
   TYPE ( HASH_control_type ) :: control
   TYPE ( HASH_inform_type ) :: inform
   INTEGER, PARAMETER :: nchar = 10
   INTEGER, PARAMETER :: length = 100
   INTEGER, PARAMETER :: new_length = 200
   INTEGER, PARAMETER :: nkeys1 = 8
   INTEGER, PARAMETER :: nkeys2 = 10
   INTEGER, PARAMETER :: nkeys3 = 3
   INTEGER, PARAMETER :: nkeys4 = 3
   INTEGER, PARAMETER :: nkeys5 = 11
   INTEGER :: i, position
   INTEGER :: MOVED_TO( length )
   CHARACTER ( LEN = 10 ) :: FIELD1( nkeys1 ) =                                &
       (/ 'ALPHA     ', 'BETA      ', 'GAMMA     ', 'DELTA     ',             &
          'X111111111', 'X111111112', 'X111111111', 'X111111114' /)
   CHARACTER ( LEN = 10 ) :: FIELD2( nkeys2 ) =                                &
       (/ 'ALPHA     ', 'BETA      ', 'GAMMA     ', 'DELTA     ',             &
          'EPSILON   ', 'X11111112 ', 'X111111113', 'X111111114',             &
          'X111111111', 'OMEGA     ' /)
   CHARACTER ( LEN = 10 ) :: FIELD3( nkeys3 ) =                                &
       (/ 'BETA      ', 'X111111112', 'OMEGA     ' /)
   CHARACTER ( LEN = 10 ) :: FIELD4( nkeys4 ) =                                &
       (/ 'OMEGA     ', 'A111111111', 'P110111111' /)
   CHARACTER ( LEN = 10 ) :: FIELD5( nkeys5 ) =                                &
       (/ 'ALPHA     ', 'BETA      ', 'GAMMA     ',                           &
          'DELTA     ', 'EPSILON   ', 'X111111112', 'X111111113',             &
          'X111111114', 'X111111111', 'OMEGA     ', 'X1111 1111' /)
!  set up the initial table
      CALL HASH_initialize( nchar, length, data, control, inform )
!  store a set of words in the table
      WRITE( 6, "( /, ' initial insertion', / )" )
      DO i = 1, nkeys1
        CALL HASH_insert( nchar, FIELD1( i ), position, data, control, inform )
        IF ( position > 0 ) THEN
          WRITE( 6, "( ' word ', A10, ' inserted  in table position ', I3 )" ) &
            FIELD1( i ), position
        ELSE
          WRITE(6, "(' word ', A10, ' already   in table position ', I3 )" )   &
            FIELD1( i ), - position
        END IF
      END DO
!  search the table for a second set of words
      DO i = 1, nkeys2
        CALL HASH_search( nchar, FIELD2( i ), position, data, control, inform )
        IF ( position > 0 ) THEN
```

```
          WRITE( 6, "( ' word ', A10, ' found     in table position ', I3 )" ) &
            FIELD2( i ), position
        ELSE
          WRITE( 6, "( ' word ', A10,' absent  from table')" ) FIELD2( i )
        END IF
      END DO
!  remove a third set of words from the table
      WRITE( 6, "( /, ' word removal', / )" )
      DO i = 1, nkeys3
        CALL HASH_remove( nchar, FIELD3( i ), position, data, control, inform )
        IF ( position > 0 ) THEN
          WRITE( 6, "( ' word ', A10,' removed from table position ', I3 )" )  &
            FIELD3( i ), position
        ELSE
          WRITE(6, "( ' word ', A10,' absent  from table' )" ) FIELD3( i )
        END IF
      END DO
!  re-search the table for the second set of words
      DO i = 1, nkeys2
        CALL HASH_search( nchar, FIELD2( i ), position, data, control, inform )
        IF ( position > 0 ) THEN
          WRITE( 6, "( ' word ', A10, ' found     in table position ', I3 )" ) &
            FIELD2( i ), position
        ELSE
          WRITE( 6, "( ' word ', A10,' absent  from table')" ) FIELD2( i )
        END IF
     END DO
!  increase the table size
      WRITE( 6, "( /, ' increase table size', / ) " )
      CALL HASH_rebuild( length, new_length, MOVED_TO, data, control, inform )
      DO i = 1, length
        IF ( MOVED_TO( i ) > 0 ) WRITE( 6, "( ' table entry in position ',     &
       &  I3, ' moved to position ', I3 )" ) i, MOVED_TO( i )
      END DO
!  store a fourth set of words in the table
      WRITE( 6, "( /, ' further insertion', / )" )
      DO i = 1, nkeys4
        CALL HASH_insert( nchar, FIELD4( i ), position, data, control, inform )
        IF ( position > 0 ) THEN
          WRITE( 6, "( ' word ', A12, ' inserted  in table position ', I3 )" ) &
            FIELD4( i ), position
        ELSE
          WRITE( 6, "( ' word ', A12, ' already   in table position ', I3 )" ) &
            FIELD4( i ), - position
        END IF
      END DO
!  re-search the table for the second set of words augmented with a further word
      DO i = 1, nkeys5
        CALL HASH_search( nchar, FIELD5( i ), position, data, control, inform )
        IF ( position > 0 ) THEN
          WRITE(6, "( ' word ', A12, ' found     in table position ', I3 )" )  &
            FIELD5( i ), position
        ELSE
          WRITE(6, "( ' word ', A12, ' absent  from table' )" ) FIELD5( i )
        END IF
```

```
      END DO
!  deallocate internal arrays
      CALL HASH_terminate( data, control, inform )
      STOP
   END PROGRAM GALAHAD_HASH_EXAMPLE
```

The code produces the following output:

```
 initial insertion

 word ALPHA      inserted  in table position  42
 word BETA       inserted  in table position  65
 word GAMMA      inserted  in table position  60
 word DELTA      inserted  in table position  56
 word X111111111 inserted  in table position  68
 word X111111112 inserted  in table position  32
 word X111111111 already   in table position  68
 word X111111114 inserted  in table position  57
 word ALPHA      found     in table position  42
 word BETA       found     in table position  65
 word GAMMA      found     in table position  60
 word DELTA      found     in table position  56
 word EPSILON    absent  from table
 word X11111112  absent  from table
 word X111111113 absent  from table
 word X111111114 found     in table position  57
 word X111111111 found     in table position  68
 word OMEGA      absent  from table

 word removal

 word BETA       removed from table position  65
 word X111111112 removed from table position  32
 word OMEGA      absent  from table
 word ALPHA      found     in table position  42
 word BETA       absent  from table
 word GAMMA      found     in table position  60
 word DELTA      found     in table position  56
 word EPSILON    absent  from table
 word X11111112  absent  from table
 word X111111113 absent  from table
 word X111111114 found     in table position  57
 word X111111111 found     in table position  68
 word OMEGA      absent  from table

 increase table size

 table entry in position  42 moved to position 157
 table entry in position  56 moved to position 130
 table entry in position  57 moved to position  86
 table entry in position  60 moved to position  43
 table entry in position  68 moved to position  90

 further insertion
```

```
word   OMEGA      inserted  in table position  57
word   A111111111 inserted  in table position   7
word   P110111111 inserted  in table position 194
word   ALPHA      found     in table position 157
word   BETA       absent  from table
word   GAMMA      found     in table position  43
word   DELTA      found     in table position 130
word   EPSILON    absent  from table
word   X111111112 absent  from table
word   X111111113 absent  from table
word   X111111114 found     in table position  86
word   X111111111 found     in table position  90
word   OMEGA      found     in table position  57
word   X1111 1111 absent  from table
```